COLLEGE CODE: 9233

COLLEGE NAME: GOVERNMENT COLLAGE OF ENGINNERING BODINAYAKANUR

DEPARTMENT:COMPUTER SCIENCE AND ENGINNERING

STUDENT NM-ID:8AE480233C809F1A2D54F980ABF4887C

ROLL NO: 923323104305

DATE: 25.09.2025

Completed the project named as Phase__2

TECHNOLOGY PROJECT NAME :REAL TIME CHAT BOT

SUBMITTED BY,

NAME :THIRUPATHI P

MOBILE NO: 9345110398

Phase 2: Solution Design & Architecture

Project Title: Real-Time AI Chatbot

Deadline: Week 7

1. Technology Stack Selection

To ensure scalability, real-time performance, and seamless user experience, the following technology stack is recommended:

➤ **Frontend**

- **React.js** or **Vue.js** – For building responsive and interactive web UIs.
- **Flutter** or **React Native** – For cross-platform mobile applications.

➤ **Backend**

- **FastAPI (Python)** or **Node.js (Express)** – For building RESTful APIs and handling real-time communication logic.

➤ **AI & NLP Engine**

- **OpenAI GPT API** – For natural language understanding and response generation.
- **spaCy / Hugging Face Transformers** – For auxiliary NLP tasks like entity extraction, summarization, etc.

➤ **Real-Time Messaging**

- **WebSockets** or **Server-Sent Events (SSE)** – To support bi-directional real-time data exchange between client and server.

➤ **Database & Storage**

- **MongoDB** or **Firebase Firestore** – For storing chat logs, user sessions, and metadata.
- **PostgreSQL** – Optional for relational data models like user accounts or analytics.

➤ **Infrastructure & Hosting**

- **AWS**, **Google Cloud Platform**, or **Microsoft Azure** – For scalable deployment, load balancing, and serverless functions (optional).

- **Redis** – For caching frequently accessed data and managing active sessions.

2. UI Structure & API Schema Design

➤ UI Structure (Web / Mobile)

Component	**Description**
Chat Window	Displays real-time incoming and outgoing messages.
Message Input Box	Allows users to type and send messages.
Conversation History	Scrollable log of past interactions, auto-loaded from the database.
User Profile & Settings	Manage preferences, language, and session data.
Typing Indicator / Loader	Shows "AI is typing..." while the response is being generated.

➤ API Schema (REST + Real-Time)

Endpoint	**Method**	**Description**
/api/sendMessage	POST	Accepts user input and initiates AI response generation.
/api/streamReply	GET	Streams AI-generated response chunks in real time (WebSocket/SSE).
/api/history	GET	Retrieves previous conversation history for the current session/user.
/api/resetChat	POST	Clears the current conversation session for a fresh start.

3. Data Handling Approach

➤ Real-Time Response Delivery

- Use **WebSockets** or **Server-Sent Events (SSE)** to send partial or streaming responses from the AI to the frontend.
- Enables token-by-token display of AI output, creating a more natural chat experience.

➤ Persistent Storage Strategy

- Store complete chat messages, timestamps, user IDs, and session identifiers in **MongoDB** or **Firebase**.
- Store structured data (e.g., feedback, user roles) in **PostgreSQL** if relational structure is needed.

➤ **Session & Context Management**

- Generate a **unique session ID** for each user conversation.
- Use **Redis** or memory cache to maintain short-term conversation context (if needed for AI continuity).

➤ **Security & Access**

- Implement **JWT-based authentication** for secure user sessions.
- Sanitize and validate all inputs to prevent XSS or injection attacks.

4. Component / Module Diagram

[User Interface (React / Flutter)]

↓

[API Gateway (FastAPI / Node.js)]

↓

[NLP Engine (OpenAI GPT / Transformers)]

↓

[Streaming Layer (WebSocket / SSE Handler)]

↓

[Database (MongoDB / PostgreSQL)]

↓

[Session / Cache Store (Redis)]

- **UI Layer** – Captures input and displays real-time responses.
- **API Layer** – Manages endpoints, routes messages, and handles auth.

- **NLP/AI Layer** – Processes input text, generates AI responses.
- **Streaming Layer** – Facilitates fast, partial data delivery.
- **Database Layer** – Persists all conversation and session data.

5. Basic Flow Diagram: Chat Lifecycle

[1] User types message ➜

[2] Message sent via /api/sendMessage (POST)

[3] Backend forwards message to AI Engine

[4] AI starts generating response (chunked)

[5] Response streamed via /api/streamReply (GET/SSE or WS)

[6] UI displays response in real time

[7] Message stored in DB with session/user info

[8] Conversation history retrievable via /api/history

Summary

This solution design offers a modular, scalable, and performance-optimized architecture for building a real-time chatbot with streaming AI capabilities. Each layer is independently deployable, which supports iterative development, testing, and scaling.

Would you like these elements converted into PDF, PowerPoint, or Diagram format (e.g., Lucidchart / Draw.io JSON) for documentation or presentation use?